GENOME-SCALE ALGORITHM DESIGN
by Veli Mäkinen, Djamal Belazzougui, Fabio Cunial and Alexandru I. Tomescu
Cambridge University Press, 2nd edition, 2023
www.genome-scale.info

## Exercises for Chapter 3. Data structures

3.1 Give an example of a perfectly balanced binary search tree storing eight (key,value) pairs in its leaves as described in Lemma 3.1. Give an example of a range minimum query for some non-empty interval.

3.2 Give a pseudocode for the algorithm to construct and initialize a balanced binary search tree given the sorted keys.

3.3 Recall how *red–black trees* work. Revisit the proof of Lemma 3,1, and consider how the tree can be maintained correctly updated for RMQ queries during the rebalancing operations needed if one adds the support for `Insert` and `Delete`.

3.4 Instead of taking the minimum among the values in Lemma 3.1 one could take a sum. If all leaves are initialized to value 1, what question does the operation analogous to RMQ answer?

3.5 Consider the sets $V'$ and $V''$ in the proof of Lemma 3.1. The subtrees rooted at nodes in $V'$ and $V''$ induce a *partitioning* of the set of characters appearing in the interval $[l..r]$ (see Figure 3.1). There are many other partitionings of $[l..r]$ induced by different subsets of nodes of the tree. Why is the one chosen in the proof the minimum size partitioning? Are there other partitionings that could give the same running time?

3.6 A van Emde Boas tree *van Emde Boas tree* (vEB tree) supports insertions, deletions, and *predecessor queries* for values in the interval $[1..n]$ in $O(\log \log n)$ time. A predecessor query returns the largest element $i'$ stored in the vEB tree smaller than query element $i$. Show how the structure can be used instead of the balanced search tree of Lemma 3.1 to solve range minimum queries for semi-infinite intervals $(-\infty..i]$.

3.7 Prove Theorem 3.3. Start as in `rank` with $O(\log^2 n)$ size blocks, but this time on arguments of $\texttt{select}_1$. Call these *source blocks* and the areas they span in bitvector $B$ *target blocks*. Define *long* target blocks so that there must be so few of those that you can afford to store all answers inside the corresponding source blocks. We are left with *short* target blocks. Apply the same idea recursively to these short blocks, adjusting the definition of a *long* target block in the second level of recursion. Then one should be left with short enough target blocks that the four Russians technique applies to compute answers in constant time. The solution to $\texttt{select}_0$ is symmetric.four Russians technique

3.8 Show how to reduce the preprocessing time in Theorems 3.2 and 3.3 from $O(n)$ to $O(n/\log n)$, by using the four Russians technique during the preprocessing.

3.9 Consider the wavelet tree in Example 3.2 Concatenate bitvectors $B_r$, $B_v$, and $B_w$. Give formulas to implement the example queries in Example 3.2 with just the concatenated bitvector. Derive the general formulas that work for any wavelet tree.

3.10 Consider the operation $\texttt{select}_c(A, j) = i$ that returns the position $i$ of the $j$th occurrence of character $c$ in string $A[1..n]$. Show that the wavelet tree with its bitvectors preprocessed for constant time $\texttt{select}_1(B, j)$ and $\texttt{select}_0(B, j)$ queries can answer $\texttt{select}_c(A, j)$ in $O(\log \sigma)$ time.

3.11 Show that the operation $\texttt{rangeList}(T, i, j, l, r)$ can be supported in $O(d \log(\sigma/d))$ time by optimizing the given search strategy. *Hint.* After finding the left-most element in the interval, go up until branching towards the second element occurs, and so on. Observe that the worst case is when the elements are equally distributed along the interval; see Section sect:kmer for an analogous analysis.

3.12 Show how to efficiently implement the operation $\texttt{rangeListExtended}(T, i, j, l, r)$ which returns not only the distinct characters from $[l..r]$ in $T[i..j]$, but also, for every such distinct character $c$, returns the pair $(\texttt{rank}_c(T, i-1), \texttt{rank}_c(T, j))$. *Hint.* Observe that the enumeration of the distinct characters uses rank queries on binary sequences that can also be used to compute the pairs of rank operations executed for the distinct characters.

3.13 In the proof of Lemma 3.17, show that insertions and deletions take expected $O(1)$ time.

3.14 In the proof of Lemma 3.18, show how to implement checking that there is no collision in (expected) linear time and linear space.

3.15 Show how to reduce the space used by the data structure from Lemma 3.19 to $O(n)$ bits while keeping the same construction and query times. *Hint.* Represent vectors $G$ and $T$ using bitvectors augmented with rank and select support.

3.16 Show how to implement an approximate membership data structure which uses space $O(n(1 + \log(1/p))$ using perfect hashing and universal hash functions. *Hint:* use the construction from the previous exercise, a universal hash function, and bit-packing.

3.17 Show that rolling Karp–Rabin hashing is universal.

3.18 Develop an alternative linear time algorithm for the sliding window minima problem discussed in the context of minimizers. In this problem, you are given an array $A[1..n]$ of values and the window length $w$, and you should compute $\operatorname{argmin}_{j \in [i..i+w-1]} A[j]$ for each $1 \leq i \leq n - w + 1$. *Hint.* When sliding the window from left to right, maintain in a linked list all pairs $(i, A[i])$ that can still become minima of some future window. The leftmost value in the list should be the minimum of the current window (essentially this list should match the rightmost path of the corresponding Cartesian tree).